Elaborar un prototipo de escritorio para reconocer imágenes de vista frontal por medio de inteligencia artificial con Python OpenCV

Universidad Tecnológica de Pereira

Rubén Darío Acuña Ocampo

C.C 1112771219

Universidad Tecnológica De Pereira

Facultad De Ingenierías

Ingeniería De Sistemas Y Computación

Pereira

2017

NOTA DE ACEPTACION

_____

_____

_____

_____

_____

_____

_____

FIRMA DIRECTOR

_____

FIRMA JURADO

AGRADECIMIENTOS

A Dios, a mi padre que se esforzó hasta sus últimos días para sacar adelante mis estudios adelante, sé que él me acompañado en espíritu los últimos semestres para sacar adelante en este proyecto.

A mi madre que me apoyo y consejos me permitió terminar los estudios profesionales, a mi hermana agradezco su apoyo y ayuda en el desarrollo de mi carrera.

A mis profesores que me enseñaron e inculcaron el amor a mi profesión.

A todos los que me apoyaron para concluir este trabajo de grado.

CONTENIDO

1. RESUMEN

El presente informe de trabajo de grado, tiene por objetivo presentar el desarrollo de una aplicación prototipo de escritorio que permita identificar una serie de imágenes preestablecida en una base de datos que permitirán a un robot previamente ya construido, tomar una decisión de avanzar, girar a la izquierda o a la derecha estas decisiones dependerán de las imágenes capturada y compararla con la base de datos para tomar la decisión correcta.

2. Introducción

Una imagen digital es una (función) f(x,y) que ha sido convertida de análoga a digital tanto en coordenadas espaciales como en luminosidad. Una imagen digital puede ser considerada como una matriz cuyos índices de renglón y columna identifican un punto (un lugar en el espacio bidimensional) en la imagen y el correspondiente valor de elemento de una matriz que identifica el nivel de gris en aquel punto. Los elementos de estos arreglos digitales son llamados pixels.

El procesamiento de imágenes es un conjunto de técnicas que permite el tratamiento y mejoramiento de la calidad de la imagen para su posterior utilización o interpretación para obtener características del fotograma como reconocer y localizar objetos en el ambiente permitiendo rastrear la posición de un objeto y determinar de manera precisa su ubicación en el ambiente, el sistema de visión depende de gran parte de seis componentes que son importantes para el funcionamiento del sistema que son los siguientes.

Captación: las imágenes son convertidas mediante la manipulación de la luz u otra forma de radiación que es emitida o reflejada en los objetos, estos reflejos de luz con captados por los sensores de la cámara, dando la forma del objeto captado y convirtiéndolo en una imagen virtual.

Procesamiento: El procesamiento o codificación interpreta los componentes de la señal de las imágenes para aplicar técnicas de reducción de ruido y realce de detalles que se pueden encontrar en el ambiente y afecten a la imagen captada.

Segmentación: Es el proceso de dividir la imagen capturada para encontrar el objeto que se desea identificar.

Descripción: El proceso mediante el cual se obtiene características convenientes para diferenciar un tipo de objeto de otro como en tamaño, color, forma.

Reconocimiento: Es el proceso de asociar un conjunto de objetos mediante color, forma y tamaño previamente ya guardados y reconocidos en una base de datos local.

Base de datos: conjunto de imágenes previamente guardadas y escogidas para la comparación de un objeto detectado con la cámara y el objeto guardado en la base de datos.

3. Definición Del Problema

Desarrollo de una aplicación prototipo de tipo escritorio utilizando el leguaje Python 2.7 y manejando la librería OpenCV que permite procesar imágenes, el sistema operativo que se utilizara en la plataforma (raspberry pi 2) contara con la distribución Linux debían raspbian jessie, la plataforma de prueba ya se encuentra previamente armada como un robot y cuenta con el siguiente hardware.

- 1 Resistencias de 220 oh
- 1 Resistencias de 470 oh
- 1 Rueda loca
- Sensor ultrasónico hc-sr04
- Puente h l293 D
- Tarjeta de red USB WIFI TL-wn725n
- PiCamera 5MP (CSI)
- 2 Servo Motores 3v
- Protoboard
- 40 Cables Jumpers
- Batería de 5v
- 2 Batería de 3.6 v
- Micro SD card 8 GB Class 10
- Raspberry pi 2
    - CPU 900MHz quad-core ARM Cortex-A7
    - 1GB RAM
    - 4 USB ports
    - 40 GPIO pins
    - Full HDMI port

- Ethernet port
- Combined 3.5mm audio jack and composite video
- Camera interface (CSI)
- Display interface (DSI)
- Micro SD card slot
- VideoCore IV 3D graphics core
- SO Linux Debian raspbian jessie

La aplicación se encargara de controlar y procesar imágenes tomadas por la cámara (PiCamera) instalada en la Raspberry pi 2 y someterla a un análisis las imágenes capturadas con el fin de extraer información de su vista frontal e identificar figuras, patrones u objetos previamente guardadas en una base de datos desarrollada para determinar qué decisión puede tomar la aplicación y enviar pulsos eléctricos a los dos servomotores que se encargaran del movimiento del robot como girar, avanzar y retroceder, estos movimientos permitirá que esquive, se detenga o se dirija a una posición determinada por las figuras que se encuentren en su entorno.

para procesar la base de datos de imágenes recolectadas y entrenar y crear un archivo capas de reconocer un objeto deseado se necesita un computador que pueda de procesar estos cálculos se utilizara el siguiente equipo:

- AMD FX 4300 Quad-Core Processor 3.80 Ghz
- RAM 12 Gb DDR3
- ASUS M5 A78L-M LX3
- HDD 2T 7200 RPM
- AMD RADEON 6670 1G GDDR3 OpenCL 1.1
- SO Linux Ubuntu Mate 16.04.3 LTS 64 Bits

4. Objetivos

4.1 Objetivo general

Elaborar un prototipo de escritorio para reconocer imágenes de vista frontal por medio de inteligencia artificial con python opencv.

4.2 Objetivos específicos

- Investigar sobre las principales técnicas de procesamiento de imágenes que se adecuen al problema.
- Identificar el dispositivo (la cámara) más adecuado para la captura de imágenes.
- Analizar la información obtenida en la investigación para seleccionar las técnicas y herramientas adecuadas para el desarrollo de la solución del problema.
- Crear el análisis y diseño arquitectónico del sistema.
- Desarrollar un prototipo que solucione el problema planteado.
- Plantear un mecanismo de prueba de errores para encontrar posibles problemas en el desarrollo de la aplicación.

5. Metodología

En el presente proyecto se utilizará la metodología de cascada que permita seguir un proceso secuencial, fácil para el desarrollo de la documentación y el prototipo, a través de fases de análisis que permitan mejorar el diseño e implantación de pruebas que determinen que errores se pueden encontrar en el desarrollo de la aplicación y la documentación, la integración y mantenimiento de los errores encontrados en fases anteriores, control en la documentación escrita durante la vida del proyecto que permita el seguimiento a través de comentarios y aprobaciones por el usuario en la gestión de la información de cada fase del proyecto.

5.1 Fases del proyecto:

- **Planeación del proyecto:** en esta fase se determinarán objetivos que se cubrirán en el trascurso del proyecto.
  - Definimos el título del proyecto.
  - Describimos la introducción del proyecto donde contextualizamos el tema en que desarrollara en el proyecto.
  - Definimos el tipo de problema que se solucionara en el desarrollo del proyecto.
  - Definir los objetivos generales y específicos del proyecto donde se investigan los métodos para el desarrollo del prototipo, los dispositivos que se necesitan usar, análisis de la información obtenida por la investigación y los mecanismos de prueba que se utilizar más adelante.
  - Definir la metodología que se utilizara en el desarrollo del proyecto.
  - Definir el cronograma de actividades donde se visualizará el promedio de tiempo del desarrollo de cada objetivo del proyecto.
  - Desarrollo de la documentación necesaria para definir los avances del proyecto.

- **Análisis:** se analiza toda la información obtenida en la investigación previa y se evalúan los requerimientos mínimos que se necesitan para el desarrollo del proyecto, con los requerimientos se puede seleccionar el hardware y software requerido por el proyecto.

- **Desarrollo del prototipo:** se desarrollará un prototipo que cumpla lo requerimientos del proyecto, estas son algunas etapas del desarrollo del prototipo:
  - Desarrollar algoritmo de detección de imágenes.
  - Desarrollar una base de datos que contenga las imágenes que detectara el algoritmo
  - Convertir la base de datos de imágenes en un archivo XML para la facilitar la interpretación de la librería opencv.
  - Implementar la base de datos XML en el algoritmo de detección de imágenes.
  - Desarrollo del algoritmo del sentido de giro de las ruedas.

- Unión de los dos algoritmos de detección de imágenes y el algoritmo del sentido de giro de la rueda.

- **Prueba de errores:** el algoritmo ya desarrollado se somete a unas pruebas exhaustivas para determinar si en el sistema no falla por algún error.

- **Verificación:** se verifica el algoritmo ya corregido por los anteriores fallos y buscar nuevas vulnerabilidades que genere el algoritmo.

- **Finalización del proyecto:** finalización del prototipo funcional y la documentación del proyecto.

6. Antecedentes

Eigenfaces es el nombre dado a un conjunto de vectores propios que se utilizan para el reconocimiento de los rostros humanos. El enfoque de usar eigenfaces para el reconocimiento fue desarrollado por Sirovich y Kirby (1987)[i] y utilizado por Matthew Turk y Alex Pentland en la clasificación de rostros. Los vectores propios se derivan de la matriz de covarianza de la distribución de probabilidad sobre el espacio vectorial de alta dimensión de las imágenes de los rostros. Los propios eigenfaces forman un conjunto de base de todas las imágenes utilizadas para construir la matriz de covarianza. Esto produce una reducción de dimensión al permitir que el conjunto más pequeño de imágenes de base represente las imágenes de entrenamiento originales. La clasificación se puede lograr comparando cómo las caras están representadas por el conjunto de bases.

El enfoque de Eigenface comenzó con una búsqueda de una representación de baja dimensión de las imágenes de la cara. Sirovich y Kirby (1987) mostraron que el análisis de los componentes principales podría utilizarse en una colección de imágenes de la cara para formar un conjunto de características de base. Estas imágenes de base, conocidas como Eigenpictures, podrían ser combinadas linealmente para reconstruir imágenes en el conjunto de entrenamiento original. Si el conjunto de entrenamiento consta de M imágenes, el análisis de componentes principales podría formar un conjunto base de N imágenes, donde N <M . El error de reconstrucción se reduce al aumentar el número de eigenpictures, sin embargo el número necesario se elige siempre menos que M. Por ejemplo, si necesita generar un número de N eigenfaces para un conjunto de entrenamiento de M imágenes de la cara, puede decir que cada imagen de la cara puede estar formada por "proporciones" de todo este K "características".

En 1991 M. Turk y A. Pentland[ii] ampliaron estos resultados y presentaron el método de Eigenface de reconocimiento facial. Además de diseñar un sistema para el reconocimiento de rostros automatizado utilizando eigenfaces, mostraron una manera de calcular los vectores propios de una matriz de covarianza de tal manera que permitan a las computadoras en ese momento realizar eigen-descomposición en un gran número de imágenes de la cara. Las imágenes de la cara usualmente ocupan un espacio de alta dimensión y el análisis de los componentes principales convencionales era intratable en tales conjuntos de datos. El documento de Turk y Pentland demostró formas de extraer los vectores propios basados en matrices dimensionadas por el número de imágenes en lugar del número de píxeles.

Una vez establecido, el método de eigenface se amplió para incluir métodos de pre procesamiento para mejorar la precisión. Múltiples enfoques múltiple también se utilizaron para construir conjuntos de eigenfaces para diferentes temas y diferentes características, como los ojos.

En 1997 B. Moghaddam y A.Pentland[iii]: realizaron un estudio que permita crear una técnica de auto aprendizaje visual que se basaba en el caculo de los espacios de una área visual. Utilizando la desintegración por medio de eigenspaces, esto permite crear una imagen lineal reducida de las caras analizadas y poderlas enfocar en un espacio reducido donde se realizará su reconocimiento. Esta técnica en su aplicación tiene más éxito que otro tipo de técnicas anteriores de reconocimiento facial ya que utiliza el cálculo de los espacio-tiempo de un área visual.

En el 2005 J. Ruiz-del-Solar y P[iv]. Navarrete realizaron estudios utilizando eigenspace para comparar algunos enfoques basados en el espacio propio. Probaron tres tipos de algoritmos para el primer método: PCA, EP y FLD. Con estos métodos diferenciales utilizaron los siguientes enfoques: el post-diferencial y pre-diferencial en ambos casos se utilizaron clasificación SVM y bayesiana, en las pruebas utilizaron la base de datos de FERET y yale facial, los resultados que obtuvieron fueron muy buenos con el enfoque post-diferencial, utilizando la clasificación bayesiana, pero tuvieron problemas con el rendimiento de procesamiento de las imágenes.

En las investigaciones de reconocimiento facial se utilizaban unas bases de datos que contenían los rostros de personas con su pelo y hombros. En el 2001 los investigadores F. Chen, H.-Y.M. Liao, J.-C. Lin y C.-C[v]. Han: se preguntaron por qué el sistema de reconocimiento facial no se enfocaba solo en la parte del rostro y descartar el pelo cuello y hombros, esto permitiría reducir el tamaño de la base de datos. Después de una serie de pruebas en diferentes bases de datos utilizando solo las caras y otra utilizando toda la imagen completa, en muchos de los resultados de las diferentes bases de datos, hay imágenes de algunas personas que no influyen mucho en los resultados que se obtuvieron, con esto se demuestra que en investigaciones anteriores que utilizaron estas bases de datos erróneas, que no se limitaban a utilizar imágenes de rostros puros como debería ser.

Hasta el año 2005 la mayoría de las investigaciones de reconocimiento facial utilizan algoritmos que usaban imágenes frontales con unas determinadas condiciones de luz. los investigadores R. Gross, S. Baker, I. Matthews y T. Kanadeeste[vi] redactaron un artículo donde plantean los problemas que pueden afectar utilizando diferentes posturas y condiciones de luz de las imágenes de los diferentes algoritmos de reconocimiento facial.

Analizaros dos algoritmos que contaban con la condición de luz como diferentes ángulos del rostro, Egen Light-Fields y Bayesian face subregions en los dos casos obtuvieron buenos resultados sin tener que restringir el número de imágenes frontales con diferentes posturas.

Uno de los últimos estudios en avances del campo de reconocimiento facial se realizó en el año 2009 L. Sirovich y M. Meytlis[vii]. Utilizaron estudios que se realzaron en la corteza de unos primates, demostraron que los impulsos neuronales en los humanos son mayores a tener un rostro completo igual a lo que ocurre a los primates, se dan cuenta que la simetría es importante,

con estos datos se crearon algoritmos de reconocimiento que permiten identificación frontal en diferentes condiciones de luz natural, solo teniendo el fallo de que si el rostro del sujeto tiene los ojos cerrados no lo reconoce.

El primer estudio científico reciente del año 2013 fue realizado en la universidad de California por Ramya Srinivasan , Amit Roychowdhury, Conrad Rudolph , Jeanette Kohl[viii] . es el primer estudio que realizan reconocimiento facial por computadora utilizando obras de arte, utilizaron obras de arte de un periodo en específico el barroco y el renacimiento ya que en esta época se tenía mucha información de los autores de las obras, el nombre de las obras y los personajes que aparecen en ellas y las técnicas que se utilizaron para pintar, en muchos retratos no se conoce la identidad del personaje y la el reconocimiento de imágenes puede dar una ayuda a buscar similitudes de estos personajes y recolectar otro tipo de datos que permitan resolver otros enigmas que tienen esta obras.

7. Ensamble del robot

- **Estructura Mecánica:** esta será como el esqueleto del robot será la encargada de dar soporte y firmeza, dicho chasis estará hecho de madera.

- **Reductores:** Para adaptar y modificar la velocidad y la fuerza del movimiento de salida de los motores hacia las llantas del robot, los motores que utilizaran trae ya incorporado una caja donde se encuentra un sistema de engranajes que permitirán realizar la reducción de velocidad y aumentando la potencia mecánica que permita mover con facilidad el peso del robot.

- **cámara:** permitirá al robot detectar imágenes ya preestablecidas en la base de datos del sistema.

- **Sistema sensorial:** para que nuestro robot puede realizar su tarea de evitar los obstáculos que encuentre, necesita hacer un reconocimiento de su entorno por lo tanto el robot tendrá un sensor ultrasónico que nos permitirán que el robot pueda detectar obstáculos que se encuentre frente a él y retroceda.

8. Materiales

Raspberry Pi: Es una computadora (SBC) de bajo costo desarrollada en Reino Unido por la fundación Raspberry Pi, con el objetivo de estimular la enseñanza de ciencias de la computación en las escuelas. En realidad, se trata de una diminuta placa base de 85 x 54 milímetros en el que se aloja un chip Broadcom BCM2835 con procesador ARM hasta a 1 GHz de velocidad, GPU VideoCore IV y 1 Gbytes de memoria RAM.

Sensor Ultrasónico: Son detectores de proximidad que trabajan libres de roces mecánicos y que detectan objetos a distancias que van desde pocos centímetros hasta varios metros. El sensor emite un sonido y mide el tiempo que la señal tarde en regresar. Estos reflejan en un objeto, el sensor recibe el eco producido y lo convierte en señales eléctricas, las cuales son elaboradas en el aparato de valoración. Estos sensores trabajan solamente en el aire, y pueden detectar objetos con diferentes formas, diferentes colores, superficies y de diferentes materiales. Los materiales puedan ser sólidos, líquidos o polvorientos, sin embargo han de ser deflectores de sonido. Los sensores trabajan según el tiempo de transcurso del eco, es decir, se valora la distancia temporal entre el impulso de emisión y el impulso de eco.

Camara pi: nos permitirá visualizar el entorno y detectar una serie de objetos que se encuentren almacenados en la base de datos del sistema.

Circuito Integrado L293D: es de gran utilidad para controlar pequeños motores y actuadores de corriente directa. Este circuito es bastante utilizado en robótica para controlar motores a pasos y de corriente directa. Incluye en su interior 4 drivers o medio puente H. La corriente máxima que el L293 puede manejar es de 1A a voltajes desde 4.5 volts a 36 volts, mientras que la corriente constante es de 600 mA.

Set de cables Macho / Hembra: Este set de cables es ideal para disponer de cables de alta calidad cuando se realiza un prototipo sobre una placa de ensayo. Todos los cables están terminados por un lado como macho y por el otro como hembra, por lo que se pueden empatar varios cables unos a los otros. El largo de cada cable es de 155mm.

Batería 3.6v: batería utilizada en los teléfonos inalámbricos, voltaje de 3.6v capacidad 550mAh.

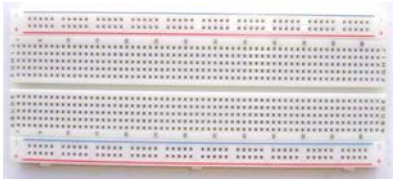
Llantas: Este par de ruedas está especialmente indicado en robots de sumo debido a su gran adherencia y su alto nivel de tracción. Esta rueda es apta para superficies tratadas de todo tipo, presentando un nivel de adherencia increíble gracias al material compuesto similar al neopreno que la forma. Su diámetro de 65 mm lo hacen perfecta para conseguir transmitir toda la potencia de los motores eléctricos.

Motorreductores: Son apropiados para el accionamiento de toda clase de máquinas y aparatos que necesitan reducir su velocidad en una forma segura y eficiente. Toda máquina cuyo movimiento sea generado por un motor necesita que la velocidad de dicho motor se adapte a la velocidad necesaria para el buen funcionamiento de la máquina. Además de esta adaptación de velocidad, se deben contemplar otros factores como la potencia mecánica a transmitir, la potencia térmica, rendimientos mecánicos. Esta adaptación se realiza generalmente con uno o varios pares de engranajes que adaptan la velocidad y potencia mecánica montados en un cuerpo compacto denominado reductor de velocidad, aunque en algunos países hispanos se le denomina caja reductora.

Chasis: Es la estructura destinaba a brindarnos la movilidad, para su construcción se debe elegir un material resistente (acrílico, madera, lámina metálica, etc.) que soporte el peso de los componentes electrónicos. El diseño del chasis determina el ancho, largo y alto del carro.

Rueda loca: Es una rueda sin tracción, simple o doble, que puede girar libremente y que generalmente está situada en la parte inferior de una estructura. Se utilizan en carros de la compra, sillas de oficinas o vehículos. Esta rueda se llama de pivote o rotatoria que se sujeta a la superficie con una estructura que tiene un eje en su centro, anclado a la rueda que puede girar libremente.

Protoboard: Es una especie de tablero con orificios, en la cual se pueden insertar componentes electrónicos y cables para armar circuitos. Como su nombre lo indica, esta tableta sirve para experimentar con circuitos electrónicos, con lo que se asegura el buen funcionamiento del mismo.



Resistencia 220 ohm: estos elementos eléctricos tiene la misión de oponerse al paso de la corriente eléctrica a través de ellas. La resistencia óhmica se mide en Ohms así como en sus dos múltiples: KiloOhm y MegaOhm. El valor resistivo es fijo. Tiene un +/- 5% tolerancia y ¼ watt.

Resistencia 470 Ohm: estas resistencias se utilizan comúnmente para distintas conexiones de LEDs de alto brillo. LEDs de alto brillo en color azul, blanco frio, blanco cálido, blanco neutro, verde, ultravioleta, rosa y cyan.

Adaptador USB para red inalámbrica: es un dispositivo portátil que tiene la función de enviar y recibir datos sin la necesidad de cables en las redes inalámbricas de área local, esto es entre redes inalámbricas de computadores. El adaptador se inserta dentro del puerto USB de la computadora y por sus características de portabilidad, no integra antena externa, sino que trae el receptor integrado dentro del cuerpo de la cubierta.

9. Implementación de los dos motores 3v a la tarjeta Raspberry Pi

Con el fin de mover nuestro vehículo impulsado por la Raspberry Pi, necesitaremos al menos dos motores de corriente continua para alimentar un tanto para la parte izquierda como la derecha de las llantas. Los motores se utilizan para mover el vehículo hacia adelante y atrás, así como la rotación de izquierda y derecha. El control de dos motores de corriente continua se llevó a cabo con un solo chip de motor, cableado y configuración del código doble puente-H L293D. Veamos el controlador de motor L293D doble puente H y el cableado GPIO. En la figura, todas las conexiones en azul y púrpura se utilizan de forma idéntica, que muestra cómo controlar un motor de corriente continua.

Se utilizan para controlar la dirección del motor 1:

- 4 GPIO
- 17 GPIO

Pulso con el control de modulación (PWM):

- 18 GPIO

Conexiones del motor de corriente continua:

- M1 +
- M1 –

Conexiones de fuente de alimentación:

- Batería +
- 5v Pi

Tierra:

- GND

Las anotaciones en verde son los que se añadió el fin de obtener el segundo motor de corriente continua para trabajar con el controlador de motor L293D doble puente H compartida.

Se utiliza para controlar la dirección del motor 1:

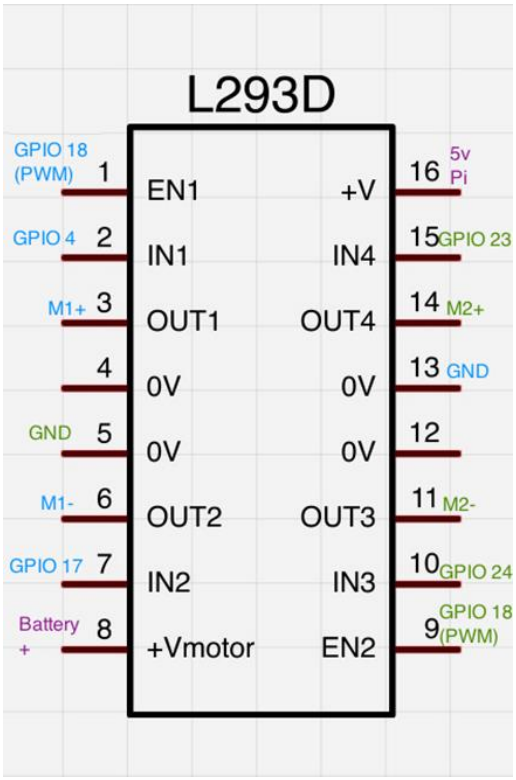
- 23 GPIO
- 24 GPIO

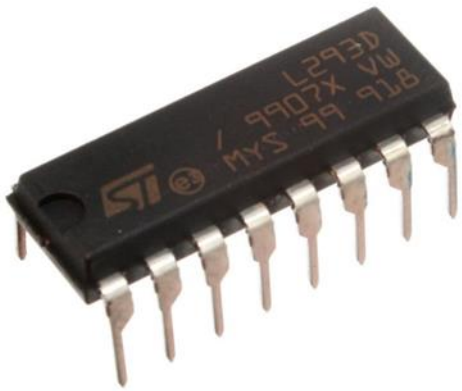
Pulso con el control de modulación (PWM):

- 18 GPIO

Conexiones del motor de corriente continua:

- M2 +
- M2

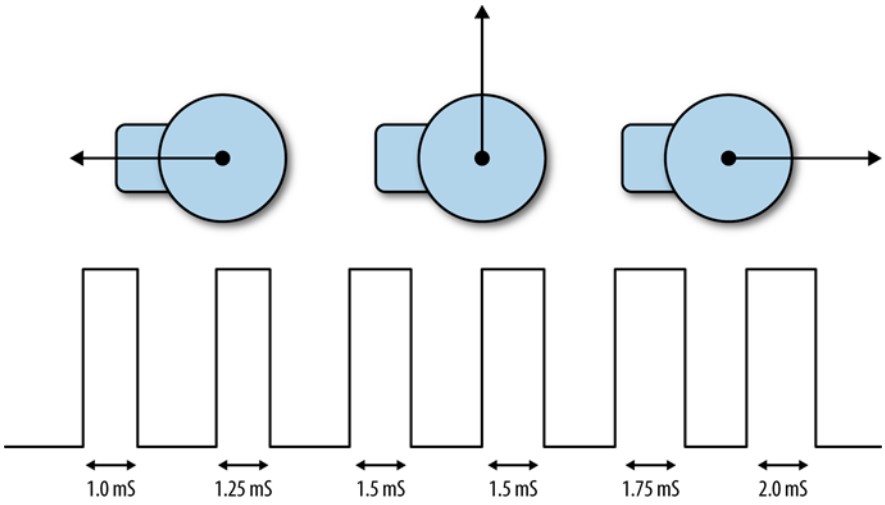




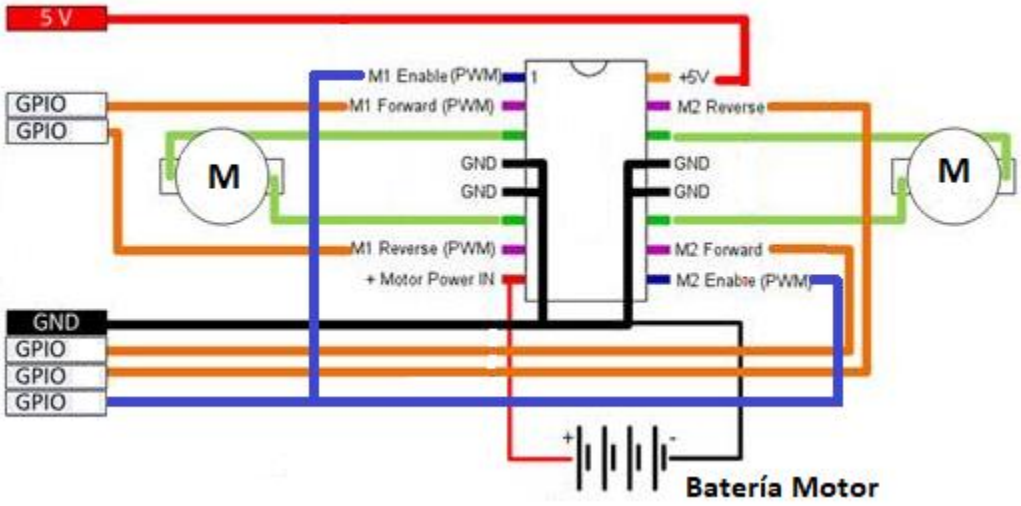
9.1 Motor PWM

En los servomotores se utilizan en vehículos de control remoto y la robótica. La mayoría de los servomotores no son continuas; es decir, que no pueden girar en todos los sentidos, sino más bien a poco más de un ángulo de aproximadamente 180 grados. La posición del motor servo se establece por la longitud de un pulso. El servo espera recibir un impulso por lo menos cada 20 milisegundos. Si ese pulso es alto durante 1 milisegundo, el ángulo del servo será cero, si es de 1,5 milisegundos, será en su posición central; y si es de 2 milisegundos, será a 180 grados. El ejemplo de programa, se establece la frecuencia PWM de 100 Hz, que enviará un pulso al servo cada 10 milisegundos. El ángulo se convierte en un ciclo de trabajo entre 0 y 100. En realidad, esto produce impulsos más cortos que el valor mínimo esperado 1 milisegundo y más largo que 2 milisegundos máximo.

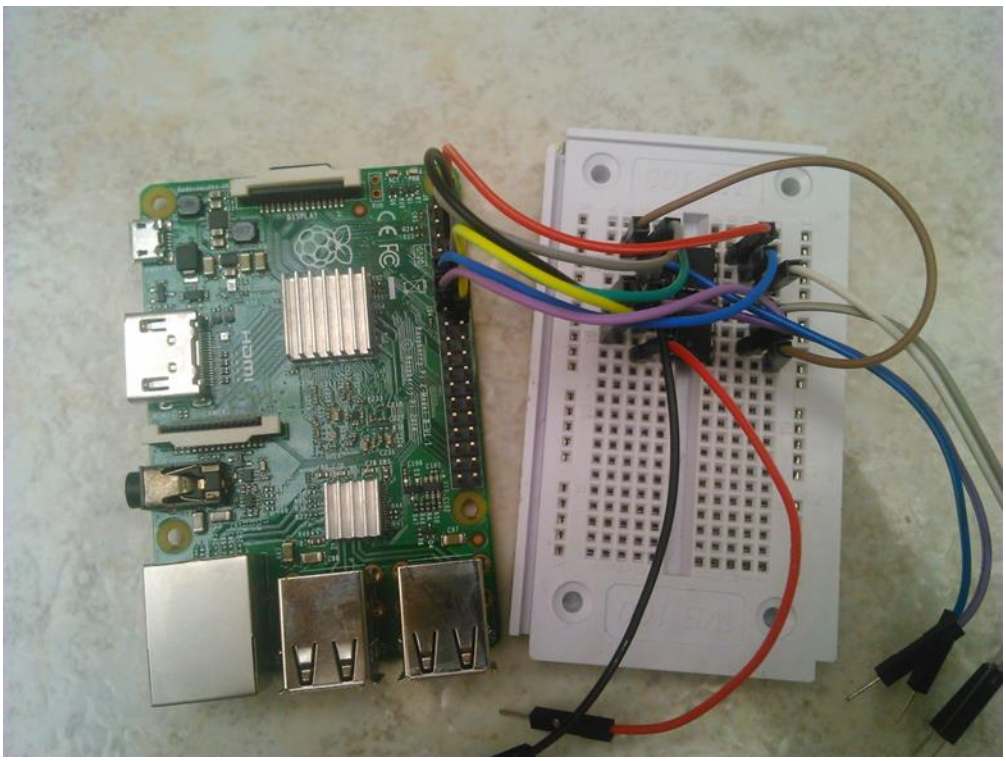
En las siguientes figuras se mostrarán como se implementan los motores 3v, el circuito integrado L293D.



9.2 Implementación de los dos motores 3v



Resultado

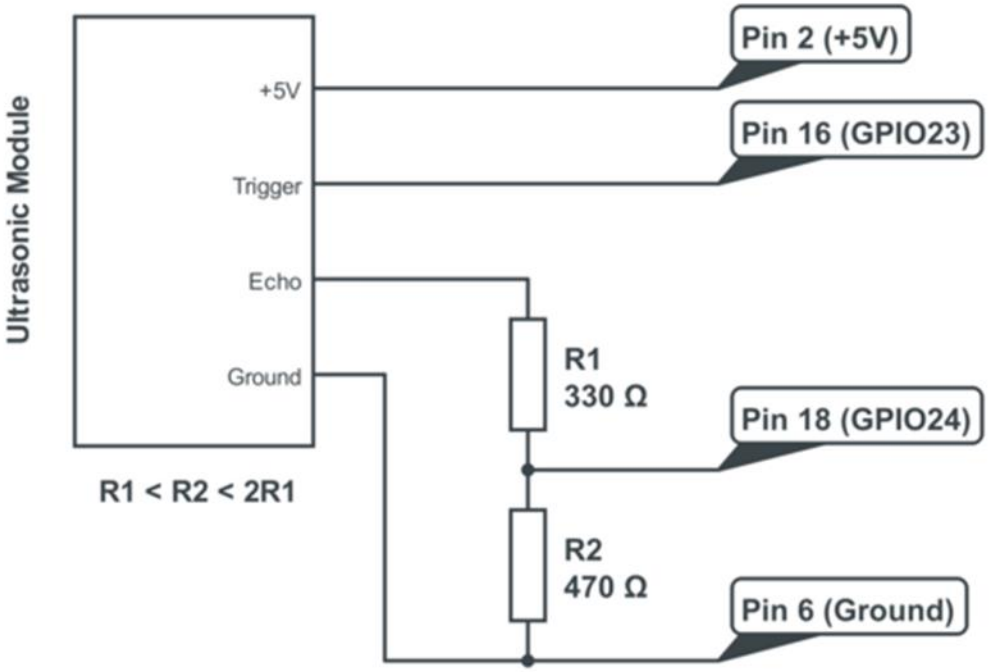


10. Implementación del sensor ultrasónico HC-SR04

El sensor ultrasónico HC-SR04 para Raspberry Pi es un sensor que, entre otras posibles aplicaciones, sirve para medir distancias. Funciona enviando un pulso de ultrasonidos a través de uno de los cilindros que componen el sensor y esperando a que dicho sonido rebote sobre un objeto y vuelva. El retorno es captado por el otro cilindro del sensor.

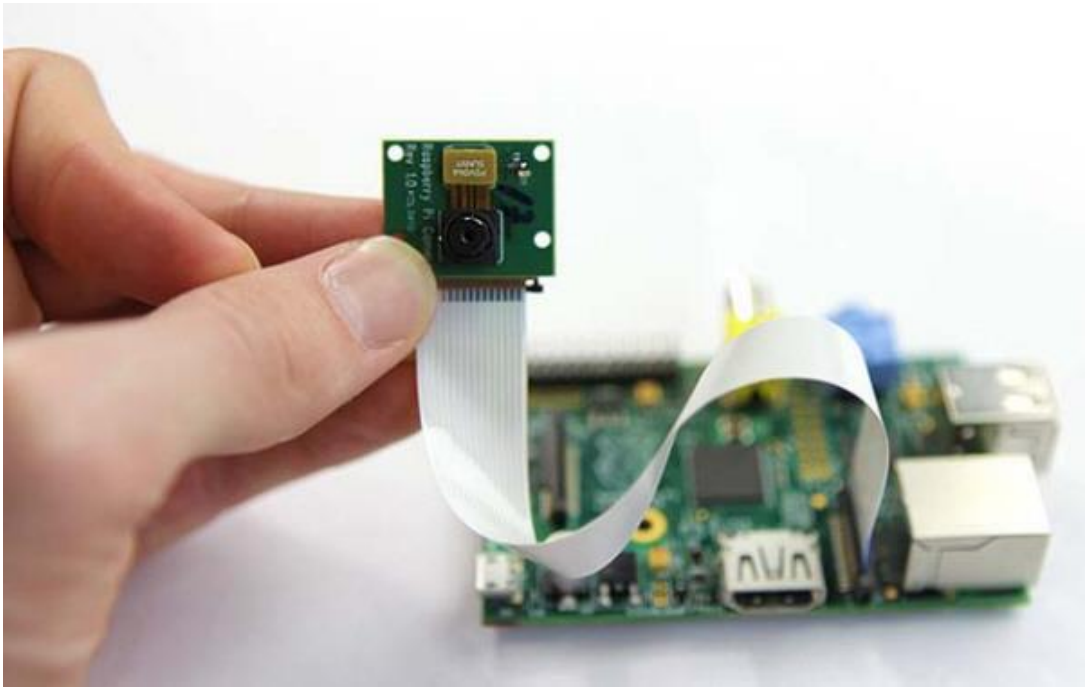
El módulo HC-SR04 es del tamaño de una caja de cerillas. Los dos transductores dan una apariencia distintiva. Está diseñado para ser accionado por 5V, tiene 1 pin de entrada y 1 pin de salida. El módulo funciona enviando un pulso ultrasónico en el aire y midiendo el tiempo que se tarda en recuperarse. Este valor puede ser utilizado para calcular la distancia recorrida el pulso.

Para alimentar el sensor se conecta la patilla VCC del módulo en el pin GIO de 5V de la Raspberry Pi y la patilla GND a un pio GIO GND, con esto ya tendríamos el sensor alimentado. Para comunicar el sensor con la raspberry hemos conectado la patilla TRIGGER al pin GIO 23 y la patilla ECHO al pin GPIO 24. Hemos puesto dos resistencia de 330 y 470 porque si R1 y R2 son los mismos entonces la tensión se divide por la mitad. Esto nos daría 2.5V. Si R2 es dos veces el valor de R1 entonces obtenemos 3.33V lo cual está bien. Así que lo ideal desea R2 sea entre R1 y R1 x 2. En el proyecto hemos usado circuito 330 y 470 ohmios. Una alternativa sería el uso de valores de 1 K y 1K5. De este modo.



11. Cámara pi

Conectamos el módulo de la cámara en el puerto CSI que se encuentra en la placa base del raspberry pi 2.



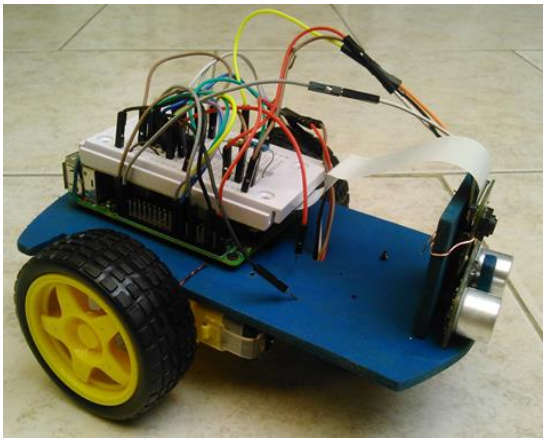
12. Alimentación al robot móvil

Para el funcionamiento de los motores se utilizamos dos baterías telefónicas de 3,6v, para el funcionamiento del raspberry pi 2 se utiliza una batería power bank de 5v a 1A de 2500 mah de capacidad.

13. Desarrollo estructura básica del robot:

- Comenzamos colocando la base de nuestro carro en la que irán anclado los motores, la rueda de balero y los demás dispositivos en el chasis del robot.
- Colocamos los motores en el chasis y los fijamos con tornillos.
- Teniendo ya los motores sujetos al chasis colocaremos el balero o rueda loca.
- Luego colocamos las llantas de nuestro robot las cuales van directamente a los motores DC.

- Ya tenemos el motor anclado en la base del chasis y ahora solo colocamos los cables en la protoboard donde está el puente H para el control de los motores, el sensor ultrasónico y la tarjeta raspberry pi.



14. Desarrollo de aplicación prototipo funcional

14.1. Herramientas de desarrollo.

OpenCV es una librería libre de visión artificial originalmente desarrollada por Intel. Desde que apareció su primera versión alfa en el mes de enero de 1999, se ha utilizado en infinidad de aplicaciones. Desde sistemas de seguridad con detección de movimiento, hasta aplicativos de control de procesos donde se requiere reconocimiento de objetos. Esto se debe a que su publicación se da bajo licencia BSD, que permite que sea usada libremente para propósitos comerciales y de investigación con las condiciones en ella expresadas.

Open CV es multiplataforma, Existiendo versiones para Linux, Mac OS X y Windows. Contiene más de 500 funciones que abarcan una gran gama de áreas en el proceso de Visión, como reconocimiento de objetos (reconocimiento facial), calibración de cámaras, visión estéreo y visión robótica.
OpenCV dispone de un conjunto muy completo de herramientas de reconocimiento y clasificación de objetos basadas en el análisis por componentes principales y el cálculo de atributos. Los clasificadores pueden ser generados automáticamente liberando al usuario de tener que especificar complejas reglas para el reconocimiento de objetos. Suministra herramientas de reconocimiento de formas capaces de procesar datos continuos, discretos e incluso simbólicos. El núcleo del módulo incluye el conjunto de clasificadores, los operadores de aprendizaje, una rutina de reagrupamiento (clustering) estándar, operadores de evaluación y análisis de atributos, así como funciones utilitarias para el PRE-procesamiento y constitución de bases de aprendizaje y la generación de clasificadores definidos por el usuario.

14.2. Instalación de OpenCV.

La instalación de OpenCV se realiza por medio de la terminal de Linux Ubuntu o debían, en este caso se instala en la raspberry pi y en el pc donde procederemos a realizar el procesamiento de las imágenes

sudo apt-get update

sudo apt-get install libopencv-dev python-opencv

sudo apt-get -f install

sudo apt-get install libopencv-dev python-opencv

algunas librerías extra que se necesitaran en el proceso:

sudo apt-get install build-essential cmake pkg-config

sudo apt-get install libjpeg8-dev libtiff4-dev libjasper-dev libpng12-dev

sudo apt-get install libgtk2.0-dev

sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libv4l-dev

sudo apt-get install libatlas-base-dev gfortran

sudo python get-pip.py

sudo pip install virtualenv virtualenvwrapper

sudo apt-get install python2.7-dev

pip install numpy

14.3. Librería de clasificación de imágenes.

El trabajo con un clasificador en cascada incluye dos etapas principales: entrenamiento y detección. La etapa de detección se describe en una documentación del módulo objdetect de la documentación general de OpenCV. La documentación proporciona información básica sobre el clasificador en cascada.

La librería que se utilizara para entrenar el clasificador de imágenes en cascada opencv_traincascade, esta librería admite las funciones Haar [Viola2001] y LBP [Liao2007] (Patrones binarios locales). Las funciones de LBP son completamente diferentes a las características de Haar, por lo que tanto el entrenamiento como la detección con LBP son varias veces más rápidas que con las funciones de Haar. Con respecto a la calidad de detección de LBP y Haar, depende de la capacitación: la calidad del conjunto de datos de capacitación, en primer lugar, y los parámetros de capacitación también. Es posible entrenar un clasificador basado en LBP que proporcionará casi la misma calidad que uno basado en Haar.

Otras utilidades auxiliares que se utilizaran en el proceso de aprendizaje es opencv_createsamples que se usa para preparar un conjunto de datos de entrenamiento de muestras positivas de prueba, opencv_createsamples produce un conjunto de datos de muestras positivas en un formato compatible con las aplicaciones opencv_haartraining y opencv_traincascade. El resultado es un archivo con extensión * .vec, es un formato binario que contiene las imágenes imágenes analizadas anteriormente. Más adelante se necesitará la utilidad opencv_createsamples para preparar los datos de entrenamiento para opencv_traincascade.

14.4. Preparación de los datos de entrenamiento.

Para entrenamiento, necesitamos un conjunto de imágenes de muestras. Hay dos tipos de muestras: negativa y positiva, las muestras negativas corresponden a imágenes sin el objeto que se desea reconocer y las muestras positivas corresponden a imágenes con objeto que se desea reconocer. El conjunto de muestras negativas debe prepararse manualmente, mientras que el conjunto de muestras positivas se crea utilizando la utilidad opencv_createsamples.

Cuántas imágenes se necesitan de muestra positiva para reconocer un objeto, los números dependen de una variedad de factores, incluida la calidad de las imágenes, el objeto que desea reconocer, el método para generar las muestras, la potencia de la CPU que tiene nuestro computador o una GPU compatible con CUDA o OpenCL y la memoria RAM disponible en nuestro equipo.

El entrenamiento de un clasificador altamente preciso requiere mucho tiempo y una gran cantidad de muestras. Los clasificadores hechos para el reconocimiento facial son excelentes ejemplos: fueron creados por investigadores con miles de buenas imágenes.

14.4.1. Muestras negativas.

Las muestras negativas se toman de imágenes arbitrarias. Estas imágenes no deben contener objetos detectados. Las muestras negativas se enumeran en un archivo especial de texto en el que cada línea contiene el nombre de archivo de imagen y la ruta donde está ubicado la imagen de muestra negativa, como el siguiente ejemplo “img/img1.jpg”.

Ahora necesitamos las imágenes negativas, las que no muestran la imagen que se desea reconocer. En el mejor de los casos, si tuviéramos que entrenar a un clasificador altamente preciso, tendríamos muchas imágenes negativas que se parecen exactamente a las positivas, excepto que no contienen el objeto que queremos reconocer.

Se utilizaron al menos 600 imágenes negativas tomadas de nuestro alrededor e imágenes sacadas de internet, para el clasificar las imágenes que se desean reconocer se utilizaron las siguientes figuras: una flecha, estrella y un corazón, se obtuvieron las imágenes por internet.

Una vez que tenemos las imágenes, las colocamos todas en una carpeta y utilizamos el siguiente comando que analiza las imágenes almacenadas y crea un archivo txt con el nombre y ubicación de todas las imágenes:

```
find ./negative_images -iname "*.jpg" > negatives.txt
```

14.4.2. Muestras positivas.

Las muestras positivas son creadas por la utilidad opencv_createsamples. Se pueden crear a partir de una sola imagen con un objeto o de una colección de imágenes recolectadas anteriormente, se necesita un gran conjunto de imágenes de muestras positivas antes de utilizar la utilidad opencv_createsamples, porque solo aplica la transformación de imágenes de una sola perspectiva. Por ejemplo, puede necesitar solo una muestra positiva para un objeto absolutamente rígido, como un logotipo de OpenCV, pero definitivamente necesita cientos e incluso miles de muestras positivas para las caras. En el caso de las caras, debes tener en cuenta todos los grupos de raza y edad, las emociones y quizás los estilos de barba.

Se utilizarán en esta muestra las siguientes imágenes y sus funciones en el robot, una pelota que al reconocerla permitirá ir hacia adelante, una estrella que al reconocerla girará a la derecha y una campana que al reconocerla girará a la izquierda.

Se necesitaron imágenes del objeto que queremos detectar, las imágenes se obtuvieron mediante Internet y se escogieron 40 de ellas, que luego podemos usar para generar muestras positivas con las que el clasificador de imágenes de OpenCV pueda trabajar.

Una vez obtenidas las imágenes, las recortamos para que solo nuestro objeto deseado sea visible. Los mejores resultados provienen de imágenes positivas que se ven exactamente como aquellas en las que desea detectar el objeto, excepto que están recortadas para que solo el objeto sea visible.

Una vez que tenemos las imágenes, las colocamos todas en una carpeta y utilizamos el siguiente comando que analiza las imágenes almacenadas y crea un archivo txt con el nombre y ubicación de todas las imágenes:

find ./positive_images -iname "*.jpg" > positives.txt

14.5. Desarrollo de muestras.

Con las imágenes positivas y negativas recolectadas, se preparan para generar muestras de ellas, que usaremos para el entrenamiento. Necesitamos muestras positivas y negativas.

Eso significa que nuestro archivo negative.txt sirve como una lista de muestras negativas. Pero aún se necesitan muestras positivas, se usar un método que no necesita una gran cantidad de imágenes positivas o negativas. Se utilizará la utilidad opencv_createsamples. Esta herramienta ofrece varias opciones sobre cómo generar muestras a partir de imágenes de entrada y nos proporciona un archivo * .vec que luego podemos usar para entrenar el clasificador de imágenes.

opencv_createsamples genera una gran cantidad de muestras positivas de las imágenes positivas obtenidas, mediante la aplicación de transformaciones y distorsiones. Como solo puede transformar tanto de una imagen hasta que ya no sea una versión diferente, necesitamos un poco de ayuda para obtener una mayor cantidad de muestras de nuestra relativamente pequeña cantidad de imágenes de entrada.

Naotoshi Seo[ix] creo algunos scripts realmente útiles que ayudan mucho al generar muestras. El primer scripts que se utilizara es createsamples.pl, un pequeño script de Perl, para obtener 1500 muestras positivas, al combinar cada imagen positiva con una imagen negativa al azar y luego ejecutarlas a través de opencv_createsamples.

Al ejecutar el scripts nos aseguramos de que estamos en el directorio raíz donde se encuentres las carpetas de imágenes positivas y negativas recolectadas y ejecutamos el siguiente comando:

perl bin/createsamples.pl positives.txt negatives.txt samples 1500\

"opencv_createsamples -bgcolor 0 -bgthresh 0 -maxxangle 1.1\

-maxyangle 1.1 maxzangle 0.5 -maxidev 40 -w 80 -h 40"

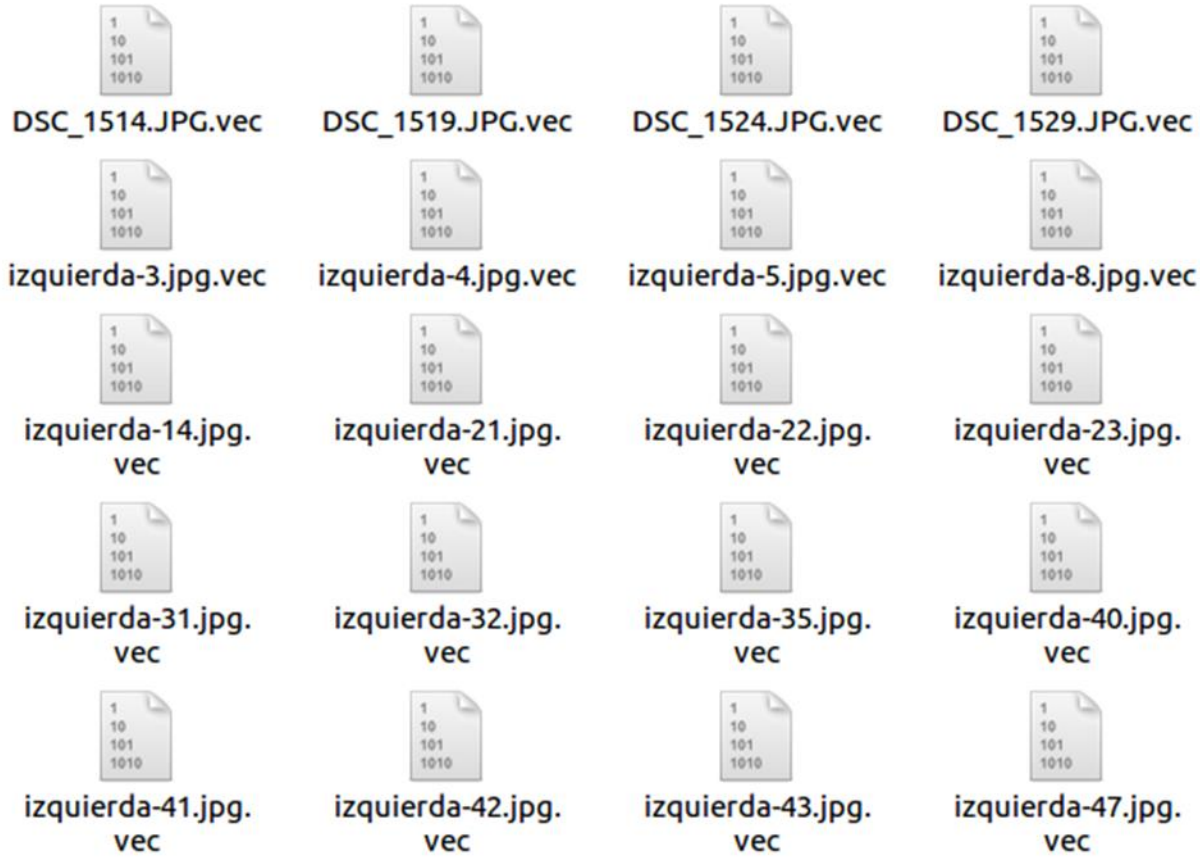
14.5.1. Argumentos de línea de comando:

NOMBRE	ARGUMENTO
-bgcolor <background_color> **-bgthresh** **<background_color_threshold>**	Color de fondo (actualmente se presuponen imágenes en escala de grises); el color de fondo denota el color transparente. Como puede haber artefactos de compresión, la cantidad de tolerancia de color puede especificarse mediante -bgthresh. Todos los píxeles con rango bgcolor-bgthresh y bgcolor + bgthresh se interpretan como transparentes.
-maxidev <max_intensity_deviation> **-maxxangle <max_x_rotation_angle>** **-maxyangle <max_y_rotation_angle>**	Desviación de intensidad máxima de píxeles en muestras de primer plano.
-maxzangle <max_z_rotation_angle>	Los ángulos de rotación máximos deben darse en radianes.
-w <sample_width>	Ancho (en píxeles) de las muestras de salida.
-h <sample_height>	Altura (en píxeles) de las muestras de salida.

Ejecución del comando:

```
opencv_createsamples -bgcolor 0 -bgthresh 0 -maxxangle 1.1   -maxyangle 1.1 maxz
angle 0.5 -maxidev 40 -w 80 -h 40 -img ./positive_images/estrella-16.jpg -bg tmp
 -vec samples/estrella-16.jpg.vec -num 54
Info file name: (NULL)
Img file name: ./positive_images/estrella-16.jpg
Vec file name: samples/estrella-16.jpg.vec
BG  file name: tmp
Num: 54
BG color: 0
BG threshold: 0
Invert: FALSE
Max intensity deviation: 40
Max x angle: 1.1
Max y angle: 1.1
Max z angle: 0.5
Show samples: FALSE
Width: 80
Height: 40
Create training samples from single image applying distortions...
Done
```

Resultado creación de los archivos positivos. Vec



El siguiente paso es fusionar los archivos * .vec que tenemos directorio raiz. Para hacerlo, necesitamos obtener una lista de ellos y luego usar la herramienta mergevec.cpp de Naotoshi Seo, para combinarlos en un archivo * .vec. para hacer el procedimiento ejecutamos el siguiente comando en el terminal:

python ./tools/mergevec.py -v samples/ -o samples.vec

14.6. Entrenar el clasificador.

OpenCV ofrece dos aplicaciones diferentes para el entrenamiento de un clasificador, que son Haar: opencv_haartraining y opencv_traincascade. Se utilizara opencv_traincascade, ya que permite que el proceso de capacitación sea de subprocesos múltiples, lo que reduce el tiempo de finalización y es compatible con la nueva API de OpenCV 2.4.9, utilizamos el siguiente comando en la terminal:

```
opencv_traincascade -data classifier -vec samples.vec -bg negatives.txt\
-numStages 20 -minHitRate 0.999 -maxFalseAlarmRate 0.5 -numPos 1000\
-numNeg 600 -w 80 -h 40 -mode ALL -precalcValBufSize 1024\
-precalcIdxBufSize 1024 -featureType LBP
```

14.6.1. Argumentos de línea de comando:

NOMBRE	ARGUMENTO
-vec <vec_file_name>	Nombre del archivo de salida que contiene las muestras positivas para el entrenamiento.
-bg <background_file_name>	Archivo de descripción de fondo: contiene una lista de imágenes que se utilizan como fondo para versiones distorsionadas al azar del objeto.
-numStages <number_of_stages>	Número de etapas en cascada para entrenar.
-minHitRate <min_hit_rate>	Velocidad de acierto mínima deseada para cada etapa del clasificador. La tasa de aciertos general se puede estimar como (min_hit_rate ^ number_of_stages).
-maxFalseAlarmRate <max_false_alarm_rate>	Velocidad de acierto mínima deseada para cada etapa del clasificador. La tasa de aciertos general se puede estimar como (min_hit_rate ^ number_of_stages).
-numPos <number_of_positive_samples>	Número de muestras positivas utilizadas en el entrenamiento para cada etapa clasificador.
-numNeg <number_of_negative_samples>	Número de muestras negativas utilizadas en el entrenamiento para cada etapa clasificador.
-w <sampleWidth> **-h <sampleHeight>**	Tamaño de las muestras de entrenamiento (en píxeles). Debe tener exactamente los mismos valores que los utilizados durante la creación de muestras de capacitación (utilidad opencv_createsamples).
-mode <BASIC (default) | CORE | ALL>	Selecciona el tipo de características de Haar utilizadas en el entrenamiento. BÁSICO usa solo

	las características verticales, mientras que TODAS usan el conjunto completo de funciones de giro vertical y de 45 grados.
-precalcValBufSize <precalculated_vals_buffer_size_in_Mb>	Tamaño del almacenamiento intermedio para valores de características precalculadas (en Mb).
-precalcIdxBufSize <precalculated_idxs_buffer_size_in_Mb>	Tamaño del búfer para índices de características precalculadas (en Mb). Cuanta más memoria tenga, más rápido será el proceso de entrenamiento
-featureType<{HAAR(default), LBP}>	Tipo de características: HAAR - características similares a Haar, LBP - patrones binarios locales.

14.7. Salida durante el entrenamiento de OpenCV.

Después de comenzar el programa de entrenamiento, imprimirá unos parámetros de entrenamiento de las imágenes positivas y negativas. Cada etapa imprimirá algunos análisis a medida que se entrena:

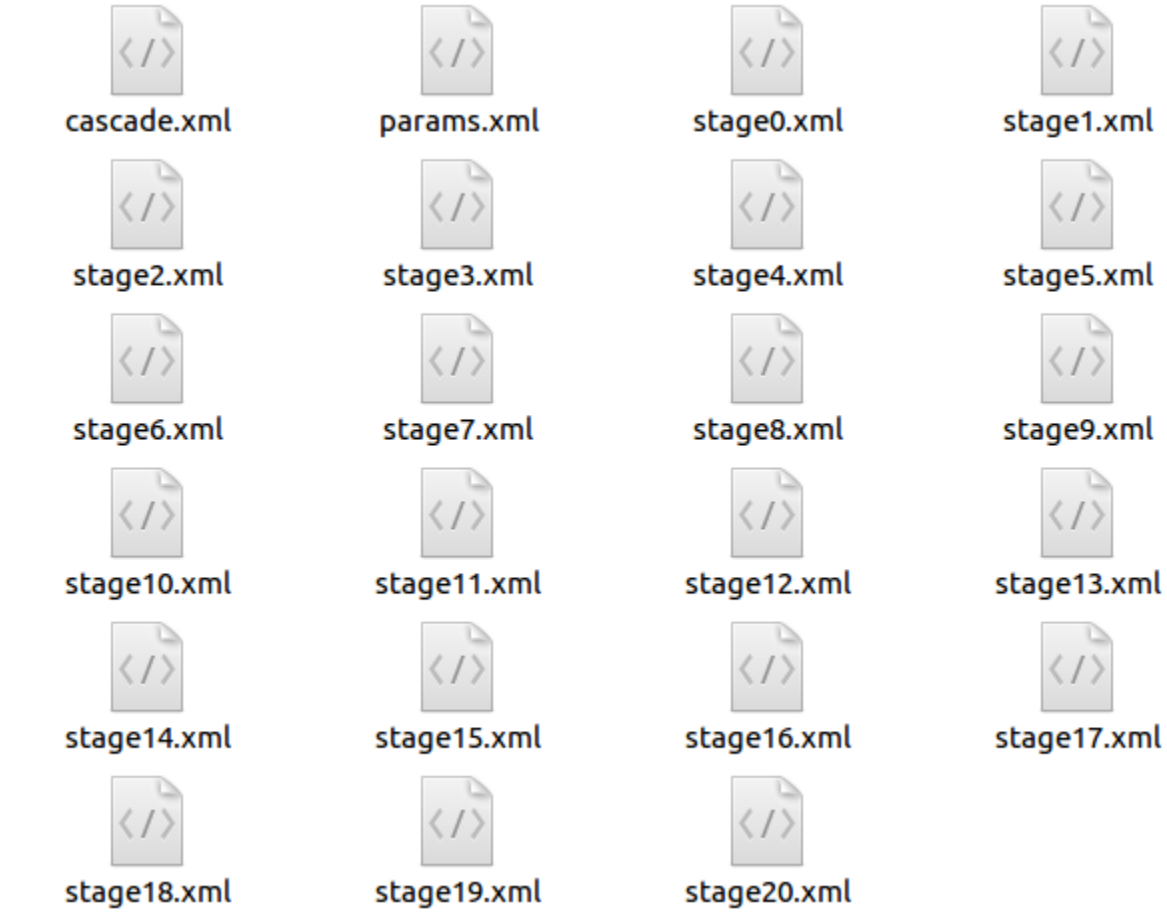
```
hinata@hinata-asus:~/Escritorio/opencv-haar-classifier-training-master$ opencv_t
raincascade -data classifier -vec samples.vec -bg negatives.txt   -numStages 22
-minHitRate 0.999 -maxFalseAlarmRate 0.5 -numPos 1100   -numNeg 900 -w 80 -h 40
-mode ALL -precalcValBufSize 1024   -precalcIdxBufSize 1024 -featureType LBP
PARAMETERS:
cascadeDirName: classifier
vecFileName: samples.vec
bgFileName: negatives.txt
numPos: 1100
numNeg: 900
numStages: 22
precalcValBufSize[Mb] : 1024
precalcIdxBufSize[Mb] : 1024
stageType: BOOST
featureType: LBP
sampleWidth: 80
sampleHeight: 40
boostType: GAB
minHitRate: 0.999
maxFalseAlarmRate: 0.5
weightTrimRate: 0.95
maxDepth: 1
maxWeakCount: 100
```

```
minHitRate: 0.999
maxFalseAlarmRate: 0.5
weightTrimRate: 0.95
maxDepth: 1
maxWeakCount: 100

===== TRAINING 0-stage =====
<BEGIN
POS count : consumed   1100 : 1100
NEG count : acceptanceRatio    900 : 1
Precalculation time: 23
+----+---------+---------+
|  N |    HR   |    FA   |
+----+---------+---------+
|   1|        1|        1|
+----+---------+---------+
|   2|        1|        1|
+----+---------+---------+
|   3|        1|     0.32|
+----+---------+---------+
END>
Training until now has taken 0 days 0 hours 5 minutes 1 seconds.
```

Cada fila representa una función que se está entrenando y contiene algunos resultados sobre su relación HitRatio y FalseAlarm. Si una etapa de entrenamiento solo selecciona algunas características, al final de cada etapa, el clasificador se guarda en un archivo xml.

Cuando el proceso finalice, encontraremos varios archivos que son los datos de cada entrenamiento y un archivo llamado cascade.xml en el directorio donde están todos los datos de entrenamiento. Este es el único archivo que guardara todas las etapas de entrenamiento anteriores.



15. Anexo código fuente

```
from picamera.array import PiRGBArray
from picamera import PiCamera
import threading
import time
import cv2
import numpy as np
import RPi.GPIO as GPIO
from Tkinter import *

#iniciar la camara
camera = PiCamera()
camera.rotation = 180
camera.resolution = (320, 240)
camera.framerate = 30
rawCapture = PiRGBArray(camera, size=(320, 240))
#inicion de base de datos de imagenes
campana_classifier = cv2.CascadeClassifier('campana0.13.xml')
circulo_classifier = cv2.CascadeClassifier('circulo0.2.xml')
estrella_classifier = cv2.CascadeClassifier('estrella0.2.xml')
t_start = time.time()

def measure():
 # calcula la distancia
```

```
 GPIO.output(GPIO_TRIGGER, True)
 time.sleep(0.00001)
 GPIO.output(GPIO_TRIGGER, False)
 time.sleep(0.00006)
 start = time.time()
 while GPIO.input(GPIO_ECHO)==0:
   start = time.time()
 stop = time.time()
 while GPIO.input(GPIO_ECHO)==1:
   stop = time.time()
 elapsed = stop-start
 distance = (elapsed * 34300)/2
 return distance
GPIO.setmode(GPIO.BCM)
# Define GPIO que se usa pi
GPIO_TRIGGER = 20
GPIO_ECHO    = 21
#motor
GPIO.setmode(GPIO.BCM)
GPIO.setup(18, GPIO.OUT)
pwm = GPIO.PWM(18,100)
left_in1_pin = 24
left_in2_pin = 23
right_in1_pin = 22
right_in2_pin = 27
```

```
class Motor(object):
    def __init__(self, in1_pin, in2_pin):
        self.in1_pin = in1_pin
        self.in2_pin = in2_pin

        GPIO.setup(self.in1_pin, GPIO.OUT)
        GPIO.setup(self.in2_pin, GPIO.OUT)

    #atras
    def clockwise(self):
        GPIO.output(self.in1_pin, True)
        GPIO.output(self.in2_pin, False)
        time.sleep(0.1)
    #adelante
    def counter_clockwise(self):
        GPIO.output(self.in1_pin, False)
        GPIO.output(self.in2_pin, True)
        time.sleep(0.2)
    #adelante
    def counter_clockwise1(self):
        GPIO.output(self.in1_pin, False)
        GPIO.output(self.in2_pin, True)
        time.sleep(1)
    #stop
    def stop(self):
```

```
        GPIO.output(self.in1_pin, False)
        GPIO.output(self.in2_pin, False)


# Establecer pines como salida y entrada
GPIO.setup(GPIO_TRIGGER,GPIO.OUT)
# Trigger
GPIO.setup(GPIO_ECHO,GPIO.IN)
# Echo
# Set trigger to False (Low)
GPIO.output(GPIO_TRIGGER, False)
time.sleep(0.01)
GPIO.output(GPIO_TRIGGER, True)
time.sleep(0.001)
GPIO.output(GPIO_TRIGGER, False)
#begin = time.time()
fps = 0
circ = 0
star = 0
campa = 0
try:
    time.sleep(2)
    direction = None
    left_motor = Motor(left_in1_pin, left_in2_pin)
    right_motor = Motor(right_in1_pin, right_in2_pin)
```

```
while True:
    for frame in camera.capture_continuous(rawCapture, format="bgr", use_video_port=True):
    # agarra la matriz NumPy sin procesar que representa la imagen, luego inicializa la marca de tiempo
        image = frame.array


    #Cargue un archivo en cascada para detectar objeto


        gray = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
    #estrella-------------------------------------------------------------
        estrellas = estrella_classifier.detectMultiScale(gray, 1.1, 5)


        print "Found "+str(len(estrellas))+" estrella(s)"
        star = int(len(estrellas))


    #Dibuja un rectángulo alrededor de cada objeto encontrado
        for (x,y,w,h) in estrellas:
            #cv2.rectangle(image,(x,y),(x+w,y+h),(255,0,0),2)
            cv2.rectangle( image, ( x, y ), ( x + w, y + h ), ( 100, 255, 100 ), 2 )
            cv2.putText( image, "estrella No." + str( len( estrellas ) ), ( x, y ), cv2.FONT_HERSHEY_SIMPLEX, 0.5, ( 0, 0, 255 ), 2 )


            if star >= 1 :
               left_motor.counter_clockwise()
               right_motor.clockwise()
               pwm.start(60)
```

```
            else :
                left_motor.stop()
                right_motor.stop()

        #campana-------------------------------------------------------------
            campanas = campana_classifier.detectMultiScale(gray, 1.1, 5)


            print "Found "+str(len(campanas))+" campana(s)"
            campa = int(len(campanas))


        #Dibuja un rectángulo alrededor de cada objeto encontrado
            for (x,y,w,h) in campanas:
                #cv2.rectangle(image,(x,y),(x+w,y+h),(255,0,0),2)
                cv2.rectangle( image, ( x, y ), ( x + w, y + h ), ( 100, 255, 100 ), 2 )
                cv2.putText( image, "campana No." + str( len( campanas ) ), ( x, y ),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, ( 0, 0, 255 ), 2 )



                if campa >= 1 :
                    left_motor.clockwise()
                    right_motor.counter_clockwise()
                    pwm.start(60)


                else :
```

```
            left_motor.stop()
            right_motor.stop()


    #circulos----------------------------------------------------------
        circulos = circulo_classifier.detectMultiScale(gray, 1.1, 5)
        print "Found "+str(len(circulos))+" circulo(s)"
        circ = int(len(circulos))

    #Dibuja un rectángulo alrededor de cada objeto encontrado
        for (x,y,w,h) in circulos:

            #cv2.rectangle(image,(x,y),(x+w,y+h),(255,0,0),2)
            cv2.rectangle( image, ( x, y ), ( x + w, y + h ), ( 100, 255, 100 ), 2 )
            cv2.putText( image, "circulo No." + str( len( circulos ) ), ( x, y ),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, ( 0, 0, 255 ), 2 )

            if circ >= 1 :

                left_motor.counter_clockwise1()
                right_motor.counter_clockwise1()
                pwm.start(80)

            else :
                left_motor.stop()
```

```
            right_motor.stop()
#print "Distance : %.1f" % distance
#sensor ultrasonico----------------------------------------------
distance = measure()
#print "Distance : %.1f" % distance
time.sleep(0.01)

if (distance <= 8 ) :
    print "atras"
    left_motor.clockwise()
    right_motor.clockwise()
    pwm.start(90)
else :
    left_motor.stop()
    right_motor.stop()

# Muestra el marco
cv2.imshow( "Frame", image )
cv2.waitKey( 1 )
# Borre la secuencia en preparación para el siguiente fotograma
rawCapture.truncate( 0 )
```

```
 #GPIO.cleanup()


except KeyboardInterrupt:
    left_motor.stop()
    right_motor.stop()
    GPIO.cleanup()
```

16. Cronograma de actividades.

ACTIVIDADES	N° de actividades				N° de semanas				N° de semanas				N° de semanas			
	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
reconocer imágenes de vista frontal por medio de inteligencia artificial																
1. planeación del proyecto																
1.1 definición del título del proyecto																
1.2 introducción																
1.3 definición del problema																
1.4 objetivos																
1.5 metodología																
1.6 cronograma																
1.7 documentación del proyecto																
2. análisis																
2.1 toma de requerimientos																
2.1.1 tipo de hardware necesario																
2.1.2 tipo de software necesario																
2.2 análisis y diseño arquitectónico																
2.3 documentación del proyecto																
3. desarrollo del prototipo																
3.1 desarrollo del algoritmo de detección de imágenes																
3.2 desarrollo de base de datos																
3.2.1 convertir la base de datos de imágenes a un archivo XML																
3.3 implementación de base de datos en el algoritmo de detección de imágenes																
3.4 desarrollo del algoritmo del sentido de giro de las ruedas																
3.5 desarrollo y unión de algoritmos para que detecte una																

figura y gire las ruedas																
3.6 documentación del proyecto																
4. prueba de errores																
4.1 mecanismo de prueba de errores del prototipo																
4.2 documentación del proyecto																
5. Verificación																
5.1 corrección de errores																
5.2 verificación del prototipo en busca de nuevos errores																
6. Finalización del proyecto																
6.1 finalización del prototipo																
6.2 finalización de la documentación																

17. Presupuesto.

Producto	**Valor Total**
1 Resistencias de 220 oh	$ 100
1 Resistencias de 470 oh	$ 100
1 rueda loca	$2.500
Sensor ultrasónico hc-sr04	$ 7.000
Puente h l293 D	$ 7.000
Tarjeta de red USB wifi	$ 35.000
PiCamera 5MP (CSI)	$ 60.000
2 Servo Motores 3v	$ 40.000
Protoboard	$ 6.000
40 Cables Jumpers	$ 10.000
Batería de 5v	$ 30.000
2 Batería de 3.6 v	$ 40.000
Micro SD card 8 GB Class 10	$ 20.000
Raspberry pi 2	$ 150.000
Total	$ 407.700

18. Bibliografía.

Raspberry (2016). https://www.raspberrypi.org/

OpenCV (2016). http://opencv.org/

OpenCV. https://docs.opencv.org/2.4.9/index.html#

Learning Image Processing with OpenCV (March 2015). Going real time: Packt Publishing Ltd. Birmingham B3 2PB, UK. Recuperado de http://opencv.org/books.html

---

[i] Sirovich, L. a. (1987). Low-dimensional procedure for the characterization of human faces. *Journal of the Optical Society of America A*, 4(3), p.519.

[ii] M. Turk and A. Pentland, "Face recognition using eigenfaces," Proc. IEEE Conference on Computer Vision and Pattern Recognition, Maui, Hawaii, 1991. (Pentland, 1991)

[iii] Moghaddam, B. and Pentland, A. (1997). Probabilistic visual learning for object representation. IEEE Transactions on Pattern Analysis and Machine Intelligence, 19(7), pp.696-710.

[iv] Ruiz-del-Solar, J. and Navarrete, P. (2005). Eigenspace-Based Face Recognition: A Comparative Study of Different Approaches. IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews), 35(3), pp.315-325.

[v] Chen, L., Liao, H., Lin, J. and Han, C. (2001). Why recognition in a statistics-based face recognition system should be based on the pure face portion: a probabilistic decision-based proof. Pattern Recognition, 34(7), pp.1393-1403.

[vi] R. Gross, S. Baker, I. Matthews, T. Kanade. (2005). Face recognition across pose and illumination. HANDBOOK OF FACE RECOGNITION (Libro).

[vii] L. Sirovich, M. Meytlis. (2009). Symmetry, probability, and recognition in face space. Proceedings of the national academy of sciences of the United States of America.

[viii] Ramya Srinivasan , Amit Roy-chowdhury . (2013). Conrad Rudolph , Jeanette Kohl. Recognizing the Royals- Leveraging Computerized Face Recognition for Identifying Subjects in Ancient Artworks. ACM.

[ix] http://note.sonots.com/SciSoftware/haartraining.html#w0a08ab4